INF122: Oblig 1

Universitetet i Bergen

2022

I denne oppgaven skal vi lage et program som analyserer og generer tekst. Tanken er å først analysere en tekst og lage en grafbasert sannsynlighetsmodell for teksten. Deretter kan man bruke denne modellen til å generere ny, tilfeldig tekst. Begge disse funksjonene skal gjøres av samme program og vi skal bruke kommandolinjeargumenter til programmet for å bestemme hvilken operasjon som skal utføres.

Markov prosesser og n-gram modeller

En **Markovprossess** en prosses som kan være i en av flere tilstander slik at i hvert steg av prossessen er det en gitt sansynlighet for at tilstanden går over i en annen tilstand. En Markov prossess kan enkelt modelleres som en graf. I grafen representerer nodene de forskjellige tilstandene og kantene er gitt et tall som sier hvor sannsynlig det er å gå fra en tilstand til en annen.

Et enkelt eksempel ville vært å modellere været som en Markov prossess. For eksempel kan tilstandene være Sol, overskyet og regn. Så kan vi tildele sannsynligheter for at været skal gå fra den ene tilstanden til en annet i løpet av en time. Se Figur 1 for et eksempel.

n-gram

Et n-gram er et begrep fra beregningsorientert lingvistikk. Det betyr ganske enkelt en delstreng med lengde n. For eksempel kan vi liste opp alle 3-gram fra strengen “haskell” som: “has”, “ask”, “ske”, “kel” og “ell”.

Alle 4-gram fra samme streng er: “hask”, “aske”, “skel” og “kell”.

Vi skal lage en Markov prossess for å generere en streng, hvor tilstandene er n-gram. Prossessen går fra et n-gram til et annet ved å fjerne et tegn fra begynnelsen og legge til et tegn på slutten av n-grammet. For eksempel kan vi gå fra “aske” til “skel” ved å fjerne ‘a’ og legge til ‘l’.

Sannsynlighene for å gå fra et n-gram til et annet får vi ved å gjøre frekvensanalyse på input-teksten. Vi skal altså itere igjennom teksten og telle hvor mange ganger hvert tegn forekommer etter hvert n-gram. Disse frekvensene lagres i en graf, som så blir vår modell for teksten.

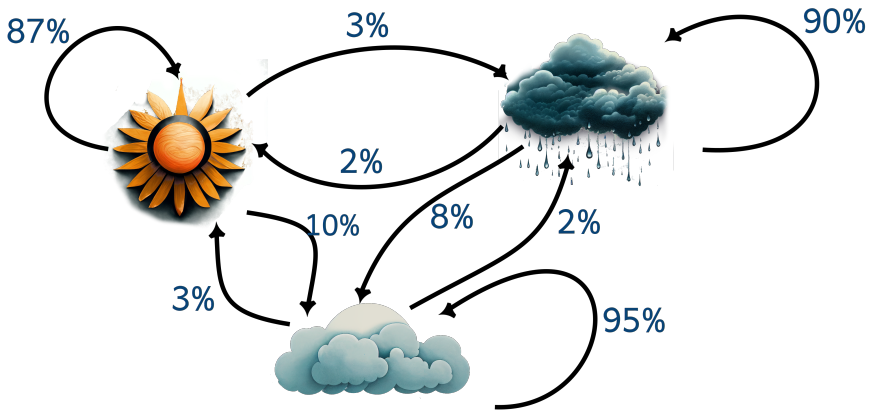

Figure 1: En Markov prossess som værmodell.

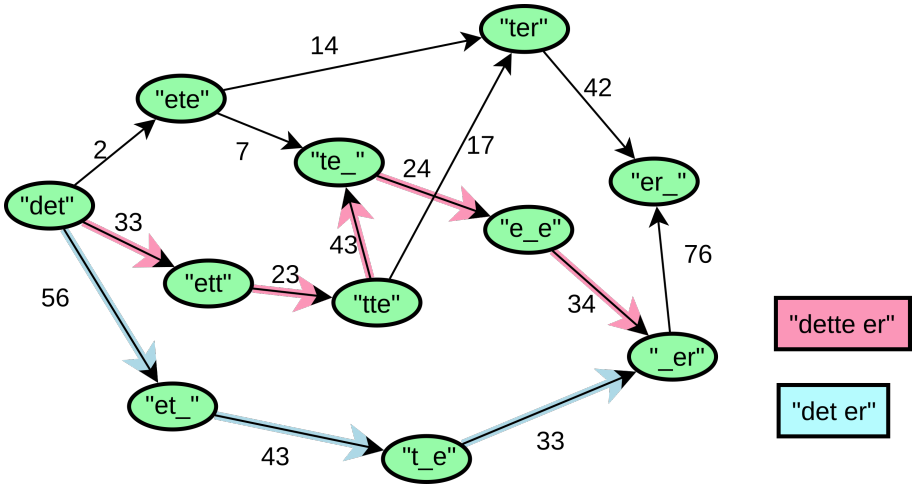

Figure 2: Et utsnitt av en 3-gram modell. De fargede pilene representerer eksempelvandinger i grafen.

Serialisering og komprimering

Det tar en stund å gå igjennom hvert n-gram av en tekst, og koden vi skriver blir nok ikke den mest effektive. Så, for å kunne generere tekst flere ganger fra samme modell skal vi lagre modellen til en fil.

For å lagre en modell til fil må vi ha en måte å gjøre den om til bytes. Map-strukturen vi bruker har allerede Show og Read instanser som konverter til og fra strenger. Vi kunne skrevet disse strengene til fil, men dette representasjonen er ganske verbos.

For å minske størrelsen skal vi komprimere teksten før den skrives til fil. Dette gjør vi relativt enkelt ved å bruke zlib bibliotekets implementasjon av GZip. Dette vil redusere filstørrelsen til ca en tiendedel.

Oppgavekoden

Koden er delt i tre filer:

- Main.hs inneholder:
  - main funksjonen som parser kommandolinjeargumenter
  - funksjoner for å lagre og laste modeller
  - funksjoner for å generere tilfeldig tekst fra en modell
- NGram.hs inneholder
  - Funksjoner for å manipulere n-gram.
- Model.hs inneholder:
  - datatypen for modeller
  - funksjoner for å oppdater og lage en modell
  - funksjoner for å hente ut en sannsynlighetsdistribusjon fra modellen

Oppgaven er avhengig av følgende bibliotek:

- zlib
- utf8-string
- random

Oppgaver

Løs hver av oppgavene under.

Oppgave 0 (3 poeng)

I denne oppgaven skal du skrive funksjoner som jobber med n-gram i NGram.hs. Husk at et n-gram er en streng med lengde n. Typen NGram er bare et annet navnt for String, men undeforstått har lengde n.

a) Skriv funksjonen `grams` som finner alle n-gram i en streng.

Fremgangsmåte: Bruk rekursjon.

Eksempel:

```
grams 3 "Haskell" == ["Has","ask","ske","kel","ell"]
grams 8 "Haskell" == []
```

b) Skriv funksjonen `gramsWithNext` som finner alle n-gram i en streng, forutenom det siste, og gir dem ut sammen med neste tegn i strengen.

Fremgangsmåte: Bruk `grams` til å få ut listen med n-gram, og bruk `zip` for å kobinere med siste char i neste n-gram.

Eksempel:

```
gramsWithNext 3 "Haskell" == [("Has",'k'),("ask",'e'),("ske",'l'),("kel",'l')]
gramsWithNext 4 "Hei" == []
gramsWithNext 3 "Hei" == []
```

c) Skriv funksjonen som rekombinerer en liste med etterfølgende n-gram tilbake til en streng:

Eksempel

```
combineGrams (grams 3 "Haskell") == "Haskell"
combineGrams ["Hei"] = "Hei"
combineGrams [] = []
```

Fremgangsmåte: Bruk patternmatching og rekursjon.

Oppgave 1 (4 poeng)

I denne oppgaven skal du skrive funksjonene for å lage og hente ut sansynligheter fra en tekstmodell i Model.hs. Typen til modellen er definert øverst i filen og består av en Map-representasjon av grafen for markovprossesen. Hver n-gram er assosiert med en tilordning av sannsynligheter for hva neste character i strengen er.

Sannsynlighetene er representert med vekter og en total vekt for hvert n-gram. Så for eksempel dersom vekten for 't' etter 3-grammet "det" er 4 av totalt 10 blir sannsynligheten 40% for at "det" følges av 't' i følge modellen.

a) Skriv funksjoen `increaseWeight :: NGram -> Char -> TextModel -> TextModel` som gitt et n-gram og neste character oppdaterer modellen ved å øke vekten for den gitte characteren etter det gitte n-grammet. Husk å også oppdatere totalvekten!

Fremgangsmåte: Bruk `Map.insertWith` for å oppdatere mappingen.

Eksempel:

```
increaseWeight "det" 't' emptyModel
==
fromList [("det",(fromList [('t',1)],1))]

let m0 = increaseWeight "det" 't' emptyModel
increaseWeight "det" 't' m0
==
fromList [("det",(fromList [('t',2)],2))]

let m1 = increaseWeight "det" 't' m0
increaseWeight "det" ' ' m1
==
fromList [("det",(fromList [(' ',1),('t',2)],3))]
```

b) Skriv funksjonen `createModel :: Integer -> String -> TextModel` som lager en n-gram modell basert på en tekst.

Fremgangsmåte: Bruk `gramsWithNext` og iterér `increaseWeight` ved hjelp av `foldl'` med utganspunkt i en tom modell.

Eksempel:

```
createModel 2 "aabaabaabaaaa"
==
fromList [
  ("aa",(fromList [('a',2),('b',3)],5)),
  ("ab",(fromList [('a',3)],3)),
  ("ba",(fromList [('a',3)],3))]

createModel 3 "Petter setter dette rett etter."
==
fromList [
  (" de",(fromList [('t',1)],1)),
  (" et",(fromList [('t',1)],1)),
  (" re",(fromList [('t',1)],1)),
  (" se",(fromList [('t',1)],1)),
  ("Pet",(fromList [('t',1)],1)),
  ("det",(fromList [('t',1)],1)),
  ("e r",(fromList [('e',1)],1)),
  ("er ",(fromList [('d',1),('s',1)],2)),
  ("ett",(fromList [(' ',1),('e',4)],5)),
  ("r d",(fromList [('e',1)],1)),
  ("r s",(fromList [('e',1)],1)),
  ("ret",(fromList [('t',1)],1)),
  ("set",(fromList [('t',1)],1)),
  ("t e",(fromList [('t',1)],1)),
  ("te ",(fromList [('r',1)],1)),
  ("ter",(fromList [(' ',2),('.',1)],3)),
  ("tt ",(fromList [('e',1)],1)),
  ("tte",(fromList [(' ',1),('r',3)],4))]
```

c) Skriv funksjoen `nextDistribution :: TextModel -> NGram -> Maybe ([(NGram, Weight)],Weight)` som slår opp et n-gram i en modell og gir en vektet liste som representerer sannsynligheten for et etterfølgende n-gram.

Fremgangsmåte: Bruk Map.lookup og Map.toList.

Eksempel:

```
nextDistribution (createModel 2 "aabaabaabaaaa") "aa"
==
Just ([("aa",2),("ab",3)],5)

nextDistribution (createModel 2 "aabaabaabaaaa") "ac"
==
Nothing
```

```
nextDistribution (createModel 3 "Petter setter dette rett etter.") "ett"
==
Just ([("tt ",1),("tte",4)],5)

nextDistribution (createModel 3 "Petter setter dette rett etter.") "tte"
==
Just ([("te ",1),("ter",3)],4)

nextDistribution (createModel 3 "Petter setter dette rett etter.") "er."
==
Nothing
```

Oppgave 2 (8 poeng)

I denne oppgaven skal du lage funksjonen som genererer tekst som følger modellen og funksjonen som laster inn en model fra en fil.

a) Skriv funksjonen `pick :: [(a,Weight)] -> Weight -> a` som velger et element fra en liste basert på en terskelverdi. Funksjonen går igjennom listen til summen av vektene av elementene den har gått igjennom overstiger terskelverdien, og returnerer elementet hvor fikk summen til å bikke over terskelverdien.

Fremgangsmåte: Bruk rekursjon.

Eksempel:

```
map (pick [("aa",2),("ab",3)]) [0..4]
==
["aa","aa","ab","ab","ab"]
```

b) Skriv funksjonen `pickRandom :: [(a,Weight)] -> Weight -> IO a` som velger et tilfeldig element fra en vektet liste med gitt totalvekt. Sannsynligheten for velge et gitt element skal være gitt av vekten til elementet.

Fremgangsmåte: Bruk funksjonen `randomRIO` til å generere et tilfeldig tall fra 0 til `total - 1` og bruk `pick` til å velge det tilsvarende elementet.

c) Skriv funksjonen `generate' : TextModel -> NGram -> Integer -> IO [NGram]` som generer en tilfeldig sekvens med n-gram gitt et start n-gram og et antall n-gram som skal genereres.

Fremgangsmåte: Bruk rekursjon. Bruk `nextDistribution` for å få en vektet liste med de neste n-grammene og bruk `pickRandom` til å velge det neste n-grammet før du rekurserer.

Eksempel:

```
let m = createModel 7 "aaaaaaaabbbbbbbb"
```

```
length <$> generate' m "aaaaaaa" 200
```

skal gi resultatet:

```
200
```

d) Skriv funksjonen `generate :: TextModel -> String -> Integer -> IO String` som gitt en vilkårlig start generer en tilfeldig streng som følger modellen. Inputstrengen skal være den første delen av outputen, og deretter fortsetter strengen etter n-grammodellen.

Fremgangsmåte: Hent ut det siste n-grammet fra startstrengen og bruk `generate'` for å generere n-gram som du så kombinerer tilbake til en streng. Husk å bruke den globale n-vedien (`gramLen`).

Eksempel:

```
let m = createModel 7 "aaaaaaaabbbbbbbb"

length <$> generate m "aaaaaaaaaaaaaaaaa" 200
```

skal gi resultatet:

```
200

generate m "aaabbbb" 7
```

skal gi resultatet:

```
"aaabbbb"
```

e) Skriv funksjonen `readModel :: Handle -> IO TextModel` som leser inn en komprimert tekstmodell fra en handle.

Fremgangsmåte: ByteString.hGetContents for å lese inn alle bytes fra filen. Bruk `GZip.decompress` for å dekomprimere dataen. Konverter bytes til strenger ved gjelp av `UTF8.toString`. Til sist kan du bruke `read` for å lage modellen fra en streng.

Testing av hele programmet: Nå er programmet ferdig og du kan teste det på noen lengre tekster:

```
$ ghc -o ramble Main.hs Model.hs NGram.hs
$ ./ramble create alice.mod alice.txt
(gå og lag deg en kopp te/kaffe)
Created model with: 85659 n-grams
$ ./ramble on "Alice went" 300
Alice went to the hedgehogs, the manage better take his head
contemptuously. "What _can_ all the eyes like after that she
had to leave their mouths. So they were all look at the Mock
Turtle replied.

There was another dead silence instantly jumped up in a
```

```
moment to stay with some children diggin
```

Du vil naturligvis få en annen generert tekst.

Å analysere en hel bok tar en stund. Hvis koden din er veldig treg kan det lønne seg å prøve på noen kortere tekster.